Methodology

We are following the Agile methodology, an iterative process in which a project is broken down into smaller individual “sprints”. Each sprint consists of 6 main stages - analyse, plan, design, build, test and review.

An Agile methodology is well-suited to our project as it focuses on close team collaboration and flexibility. Being flexible allows us to make improvements as we develop and thus will increase the quality of our overall finished product, and working together closely means that if one person is unable to complete one task, others can account for them. Agile methodology also allows for a lot of customer involvement, ensuring that the game is accurate to what the client wants at every step in the process. In our use case, we set sprints after every weekly team meeting, allowing us to discuss and set manageable deadlines for each. Below is a general breakdown of how we structured our sprints.

Analyse

Here we would identify what needs to be done, clear requirements and any risks that could be faced along the way.

Plan

Then it would be decided which deliverable team would meet which requirements for the week. For larger requirements that covered multiple deliverables, these would be broken down into smaller requirements & designated to each team.

Design

Here, we created prototypes to see how the features work in practice. These prototypes are useful as they allow us to have a sort of “practice run”, meaning our final code is better designed, and to see a feature in action before dedicating too much time. We would also create mockups of designs, for instance, user interfaces, at this stage.

Build

After this, we move on to the build stage of the sprint, where we write and implement the code and features needed. During this, we use either pair programming or simply reviewing each other's code to ensure only high-quality code is built at each stage. After building, we have working software and up-to-date documentation.

Test

With Continuous Integration, this stage would run parallel to the building process, meaning the program is never in an unusable state week by week. The use of CI ensured that the program is always in a workable state, and that branches aren’t too far separated from the current build - this is especially important in our agile methodology, as we often change and pivot when needed.

Review

Finally, we come together and review what went poorly and what we could do to

improve this in the next sprint cycle. This naturally meant that our workflow was constantly evolving as we learn throughout the course of the project.

Tools

In terms of collaborative tools, our team has made use of GitHub for version control and branch management of our code. It is a useful tool which allows us to safely commit and work on separate features at the same time, ideal for our agile methodology and for working with each other in a team. We have also used Google Drive and Docs to share planning and documentation, which is convenient because we can all access this information at any time and even collaborate directly while writing. WhatsApp was also used for communication; its availability on smartphones meant that it allowed for quicker lines of communication outside of meetings. Trello was used as an informal planning tool to arrange task priorities and their dependencies quickly. Being web-based meant individuals could stay updated on tasks they and others were working on. Due to the nature of its use, alternatives were not considered, as its use did not underpin the majority of the project progress. For a more formal representation of tasks, PlantUML was used to create Gantt charts, as well as other UML diagrams, in the documentation. It was chosen over other alternatives due to its ability to be accessed offline and create non-UML diagrams, such as Gantt charts. However, due to a feature discovered later in the project in IntelliJ, many UML diagrams could be created automatically.

In terms of development tools, both IntelliJ & Visual Studio were used by the implementation team. Their compatibility with GitHub meant that we could use its built-in CI feature, GitHub Actions, increasing the efficiency of the testing phase of each sprint. Also, both had built-in stylechecking, also aiding in CI. Photoshop was chosen for sprite design due to our game artist's prior experience and familiarity, as well as the efficiencies provided by its tool range, e.g., the Magic Wand, which allows for quick recolouring and adjustment of assets.

Organisational Structure

Upon taking over the previous team’s project, it was quickly realised that the previous team’s structure would not fit ours; this point is elaborated on in the change report.

Within the first few meetings, the decision was made to allocate team members to specific roles based on each deliverable. The number of people per deliverable was mostly based on the weighting of marks & and an effort was made to ensure that each team member contributed approximately 13 marks' worth of work. However, no team consisted of just one person, in a conscious effort to reduce risk. Additionally, art and design roles were created for both GUI & assets, along with a team/scrum leader role. Role allocation prioritised experience, as it meant less time was needed learning skills in a time when progress could

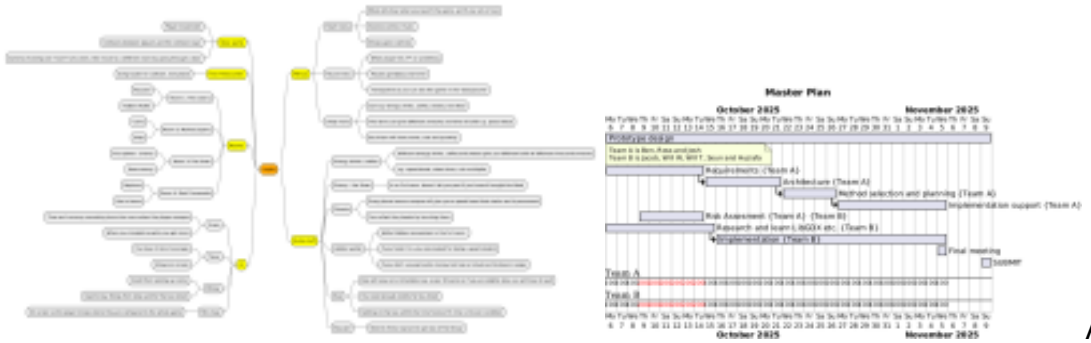
be made. This meant that if a member was on a team for a deliverable in the previous assessment, then they would be involved with that portion of the changelog or continue as part of the implementation team. After those experiences in a role were assigned, gaps in teams were then filled based on people's preferences. All roles were recorded on a Google Sheet. Below is a chart of each member's roles.

Team Member	Role	Team Member	Role
Will King	User Evaluation Art & Design (Assets)	Hari Thorman	Change report - Requirements Testing - Report
Charlie Thoo-Tinsley	Change report - Risk assessment Implementation - Docs User Evaluation	Aiden Turner	Implementation - Code Testing - Code & Report
Euan Cottam	Change report - Method selection CI Report - Report	Marcus Williamson	Implementation - Code CI Report - Github Infrastructure
Zach Moussallati	Change report - Intro	Joshua Zacek	Website Change report - Architecture Testing - Website Art & Design (GUI) Scrum Leader

Systematic Plan

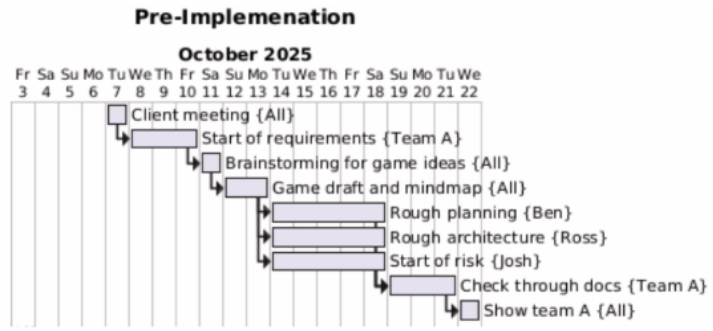
Assessment 1

After our meeting with the client, we started the requirements doc, getting a list of everything the client has requested. We then had a brainstorming session, getting a lot of ideas from everyone in the group. We wrote everything down, then went back through the list and eliminated all the ideas that didn't fit the user requirements, we as a group didn't think were interesting enough to make into a full game, and just didn't seem fun enough for us to code. Now the list was much smaller, we did a rough drawn draft of what each idea would look like. We very quickly unanimously decided on the idea we have got now. At the next meeting, we developed the basic idea fully, created a doc of it all, then we put it in a mindmap found below.

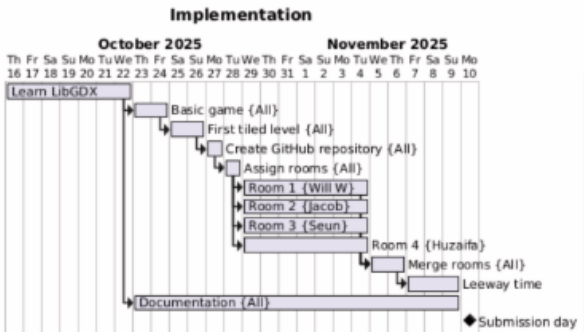


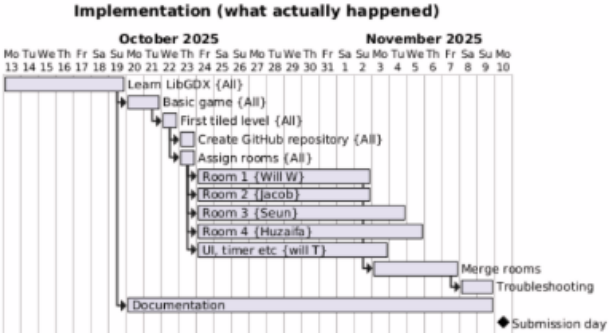
After the rough plan, we made a Gantt chart, with a rough plan with each deliverable, how long we thought it would take, and what needs to be done before the next.

We quickly realised this is not accurate of how we are going to be doing this project as most deliverables can't be fully completed before others, so we changed it. We needed to split each deliverable into smaller parts, but we often came back to them and did a bit more. Within the group, we worked out which bits that are part of team A's job needed to be done for team B to start coding the game (e.g. the start of requirements and risk, and a rough draft of planning and architecture) but we didn't need for anything to be perfect or fully complete yet as we knew once the coding team has started, we would have plenty of time to go back and turn our rough documentation into clean, well-written deliverables. Once we had a list of what needed to be done first, we put it in a Gantt chart with dependencies for team A to do.



We finished these a few days earlier then expected so we could get an earlier start on implementation. We created a Gantt chart of what team B needs to do to code the game for assessment 1. On the left was our original plan, but it changed over the weeks to be the chart on the right.





This changed timings of things is to be expected, as we didn't correctly anticipate the amount of code for individual rooms as well as how quickly team A finished the required documents as shown above. Merging rooms took more time then expected, but we had built in leeway time for us to be fine. Ideally we wouldn't be working this close to the deadline, but we have gotten it all done in time. If we were to do this again, we would definitely create more leeway into it, giving more time for accidents/mistakes to happen.

Whilst team B are busy coding, team A are busy with finishing the documentation. We have a good start on lots of it, but all of it needs refining and completing fully. We decided for our team a weekly Gantt chart would be better for us to break all the tasks into smaller, more manageable chunks. This allows us to easily give some tasks more time if they are taking more time than expected, and builds in a bus factor from if anything happens, all of us on the team have been caught up to date on what they were doing, so anyone can jump in and

take over if needed. This worked really well, and we will be doing this way of working for the next assessment.

Throughout the whole project, we have been using Whatsapp chats to keep in contact, having a whole team chat, and separate ones for the two teams. This allows for easy communication between us, organising meetings, any troubleshooting help, etc. We also have a google doc for each team, which for team A has a to-do list, and if anything we think of that needs to be added, we write it down on there. And for team B, theirs has each person's coding responsibilities, as well as common variable names, etc.

During the implementation of the game, our strategy of splitting up team A so everyone had their own bit to code meant each room had lots of detail in the code and things within it, however, we left merging the rooms to the end of development, so there was quite a lot of troubleshooting and editing of peoples code so it is still modular and easy to add more rooms in the future. We have got it working now, but in the future we would definitely start coding with a stronger base in the sense of modularity, e.g. having room by room draw and movement functions. This would allow for easier modularity, since anyone could code anything within their rooms code, and it would be runnable by the main code.

Assessment 2

For this project, one of the key problems that arose in planning the development timeline was that there would be a period over Christmas during which most team members would be geographically separated, making physical meetings unrealistic. For this reason, the choice was made to split development into three separate stages. Below is a breakdown of all three stages of development.

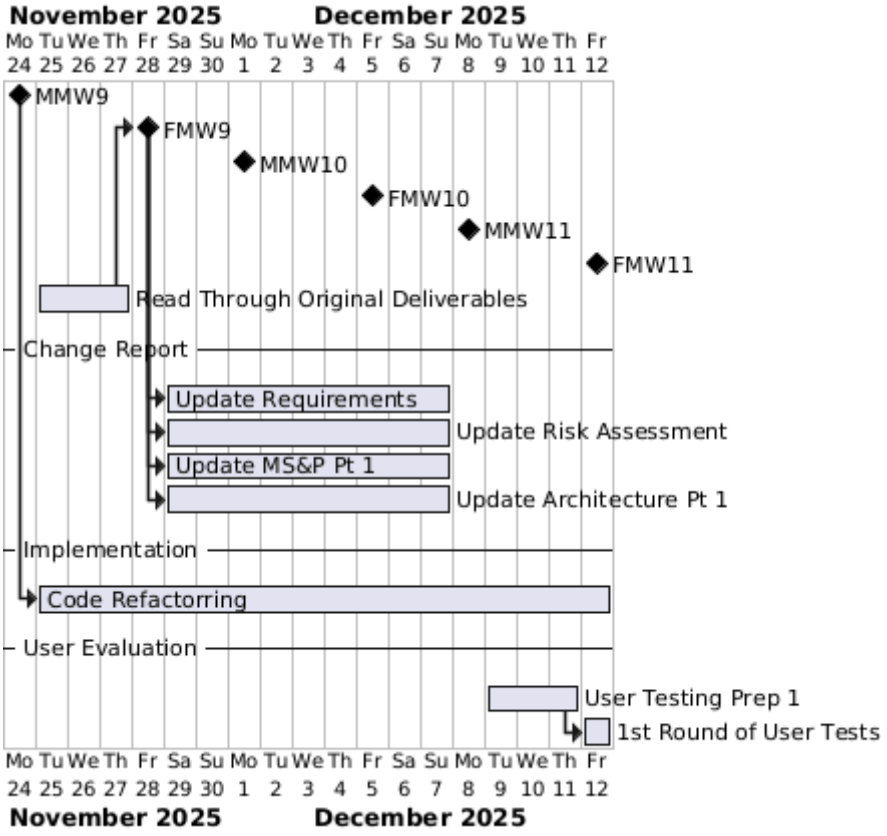
Part 1 - Change Report, User Evaluation & Refactoring of Code

For the change report team, it was expected that all members would review their respective deliverables & make notes on the state of each. Then, at the group meeting on Friday during week 9, these changes would be discussed. Then over the next week & a half, all members were expected to make these changes & note & justify them on a shared change report document. However, these plans did change when it came to our Monday meeting in week 11, as implementation was dependent on requirements, this was completed along with risk assessments, but method selection & planning was partially completed with the systematic plan write-up being moved to part 3, so that it could be written with a perspective of the entire project & space was set aside to add to architecture in case architecture of the final product was changed over part 2 in response to any problems arising in the implementation of it.

In parallel to the change reports, the implementation team was unable to make further progress on the product because it depended on updated requirements. Therefore, in the meantime, they spent time refactoring code to make it easier to extend the previous team’s work.

In week 11, as the first round of user testing was underway, the Monday meeting was spent organising & preparing for the user tests. Then, for the rest of the week, the user testing team were expected to prepare for the first round of testing, which was scheduled for Friday.

During part 1, weekly meetings would be held twice a week, and work discussions would take place during those meetings. Below is a Gantt chart used for part 1 of the project.



Part 2 - Implementation, Continuous Integration & Testing

During part 2 of the project, most members would be away from the university for at least three weeks. This meant that we had to prepare for a more remote working environment. We had planned two remote meetings for 19/12/2025 & 04/01/2026 to keep team members in the loop.

Most of this part involved the implementation team extending the project. During the first meeting, a shared Trello page was created to track tasks. Along with this, our continuous integration methods & unit testing were created & implemented.

Part 3 - Finalisation, Website & Final Documentation

The time allotted for part of the project was only one week. During this time, it was expected that all team members to finish all final documentation within the week, all culminating in a longer final meeting on the final Sunday before submission. Additionally, the final round of user testing took place on Monday of this week.

On Sunday, all team members came together & proofread all finalised deliverables & final last-minute changes were made to fix all bugs in the final product. By the end of the meeting, all deliverables were built & collated, ready for submission.